Declaration of Authorship

This is to certify that the work done by the students listed below was done under the supervision of Mr. Ahmed Shohel, Assistant Professor at the Islamic University of Technology's Department of Computer Science and Engineering (IUT). This article is the result of the student's thesis work for the Bachelor of Engineering in Computer Science.

Author: Ahmed Camara
ID: 170041070
E-mail: camaraahmed@iut-dhaka.edu

14/05/22 .....
Date and Signature

Author: Aanmar Abdou Salam
ID:170041075
E-mail: abdousalam@iut-dhaka.edu

14/05/22.....
Date and Signature

Author: Mefire Abdallah
ID:170041078
E-mail: mefireabdallah@iut-dhaka.edu

14/05/22.....
Date and Signature

Malware Detection Using Machine Learning Classifiers

Approved By:

Supervisor:

Shohel Ahmed

Assistant Professor

Department of Computer science and Engineering (CSE)

Islamic University of Technology (IUT), OIC

turnitin

Similarity Report ID: oid:22779:17161214

PAPER NAME

Report.pdf

WORD COUNT

3837 Words

CHARACTER COUNT

21566 Characters

PAGE COUNT

23 Pages

FILE SIZE

563.9KB

SUBMISSION DATE

May 13, 2022 10:50 AM GMT+6

REPORT DATE

May 13, 2022 10:50 AM GMT+6

● 30% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 19% Internet database
- Crossref database
- 24% Submitted Works database
- 19% Publications database
- Crossref Posted Content database

16/05/2022

Summary

ISLAMIC UNIVERSITY OF TECHNOLOGY (IUT)

Malware Detection Using Machine Learning Classifiers

Authors:

Ahmed Camara: 170041070
Aanmar Abdou Salam: 170041075
Mefire Abdallah: 170041078

Supervisor:

Shohel Ahmed
Assistant professor
CSE Department, IUT

A thesis submitted in partial fulfillment of the requirements for the degree of B.Sc. Engineering in Computer Science and Engineering

Academic Year 2020-21

Department of Computer Science and Engineering (CSE)
Islamic university of Technology (IUT)
A Subsidiary Organ of Organization of Islamic Cooperation (OIC)
Board Bazar, Gazipur-1704, Bangladesh.

May, 2022

Declaration of Authorship

This is to certify that the work done by the students listed below was done under the supervision of Mr. Ahmed Shohel, Assistant Professor at the Islamic University of Technology's Department of Computer Science and Engineering (IUT). This article is the result of the student's thesis work for the Bachelor of Engineering in Computer Science.

Author: Ahmed Camara
ID: 170041070
E-mail: camaraahmed@iut-dhaka.edu

..……..
Date and Signature

Author: Aanmar Abdou Salam
ID:170041075
E-mail: abdousalam@iut-dhaka.edu

……..
Date and Signature

Author: Mefire Abdallah
ID:170041078
E-mail: mefireabdallah@iut-dhaka.edu

……..
Date and Signature

Malware Detection Using Machine Learning Classifiers

Approved By:

Supervisor:

Shohel Ahmed

Assistant Professor

Department of Computer science and Engineering (CSE)

Islamic University of Technology (IUT), OIC

Acknowledgment

We would like to express our sincere gratitude to the Computer Science and Engineering Faculty for allowing us to complete this thesis, as well as to our supervisor, Mr. Ahmed Shohel, Assistant Professor. For this thesis, his explanations and ideas were invaluable. Without his leadership, none of this would have been possible. From the initial phases of the work and topic selection to project implementation and finalization. His important opinions, times, and inputs were offered throughout the thesis work, which aided us in completing our thesis work properly. Mr. Ahmed Shohel's suggestions and comments on our work were really appreciated.

Abstract

With the growth of technology, and the exponential amount of data that is being generated, the main challenge is to figure out how to protect this data from unauthorized access. Over the last couple of years, researchers have struggled to come up with a best solution that would handle this problem. The signature-based detection was the standard method used to detect malware. Regrettably, traditional technologies are no longer capable of providing adequate protection. In this work, we proposed a protection system where we trained different models in machine learning to learn from malicious and benign files to allow future prediction. We trained three classifiers in this work, Random Forest, Decision Tree, and KNearest Neighbors on the data. Random Forest gives the best result with an FPR value of 0.0208 and an accuracy of 98%.

Keywords: Malware, Machine Learning, Random Forest, Decision Tree, K-NN, Sampling

Contents

List of Figures

1. Introduction

1.1 Overview

We live in a world where computer devices have become much more necessary for our daily life. The computer is being used by everyone starting from normal users to engineers, which leads to an exponential amount of data that is being generated. Therefore, securing these data has been one of the most challenges for engineers over the past decades. Many techniques were used to come up with the best solution. Various solutions have emerged starting from traditional solutions to the most advanced ones. According to the independent IT-Security institution [1], the number of malware has exponentially increased over the last couple of years, going from 182.90M in 2013 to 1347.63M in 2022. The term “Malware” [2] stands for malicious software that is used by cybercriminals to access unauthorized data. There are different types of malware such as Ransomware, Trojan, Spyware, etc[3]. The standard method that was used was known as signature-based detection[4] which is the most popular technique used to detect malware. But as the number of malware keeps increasing exponentially, this method has become useless, because signature-based detection can only detect already seen malware, not the unseen ones. There are also different types of techniques to detect malware such as static analysis [5] which consists of analyzing a malware source without running it, and dynamic analysis [6] consists of running the malware in a virtual environment to analyze the behavior. Many efforts have been made to implement an advanced technique to detect malware.

1.2 Problem Statement

The signature-based detection is a technique where, when new malware is released on the market, the anti-malware company has to analyze the newly released malware, assign a unique string identifier to the malware, and update the anti-malware database to allow future detection. Between this period of analysis and implementation of the solution, thousands or millions of computers might get infected. This makes clear how inaccurate signature-based detection has become. We need a technique that can overcome this problem. This is where Machine

Learning classifiers come into play. Machine Learning allows us to train a model that will be able to recognize malware and benign files and make a prediction whether a newly installed file in a system is legitimate or not.

2. Literature Review

Zane Markel and Michael Bilzor [7] proposed a binary classification to detect malware. In this paper, researchers extracted a set of features from the PE header sections of legitimate, and non-legitimate files. In this experiment, they used Logistic Regression, Decision Tree, and Naive Baye. The result showed that Decision Tree gave the best result with an F1-Score of 0.97.

Junho Choi, Hayoung Kim, Chang Choi, and Pankoo Kim [8] proposed a method by extracting features using N-gram analysis from a malicious code for classifying executable as malware or benign. They used Support Vector Machines (SVM), and K-Nearest Neighbors (KNN) in this experiment. The result showed that KNN outperformed SVM with an FPR value of 0.009, and a result of 0.015 for SVM.

Zhongru Wang, Peixin Cong, and Weiqiang Yu [9] proposed three approaches which are: The Extraction and validity verification methods of malicious code metadata based on the big data analysis framework Spark, the extraction and learning of metadata of PE- files are more efficient so that PE- Classifier can process large-scale PE files distributed and then Propose and implement the malicious code detection prototype PE-Classifier for quick and distributed detection with great accuracy compared to the traditional AV detection where the great result has been founded. the evaluation indicator for EP- classifier shows the result of 0.96 for TPR.

Ivan Firdausi, Charles Lim, Alva Erwin, and Anto Satriyo Nugroho [10] in their work, have extracted a set of features from the Window Portable Executable(PE) files. using a total of 220 unique malware samples. The benign were taken from system files located in the “System32” directory of a clean installation of Windows XP Professional 32-bit with a total of 250 unique benign software samples. In this work, they trained 5 classifiers KNN, Naive Bayes, SVM, and J48 Decision Tree.

The experiments were conducted based on four data sets. J48 Decison Tree outperformed with an accuracy of 95.9%, and an FPR value of 0.024.

Athiq Reheman Mohammed (&), G. Sai Viswanath, K. Sai babu, and T. Anuradha [11], have also proposed a technique where they extract a set of features from the PE header file to construct a dataset with the respective values, and then train many models to learn from this data and make predictions. To test the model, they created 9 a static website where they uploaded a file, and at the back-end side, the trained machine learning model classifies whether the uploaded file is malicious or not. They got the best accuracy with Random Forest, at 99.4%. In a similar approach, Nur Syuhada Selamat, Fakariah Hani Mohd Ali [12], in their work performed analyzed malware using static analysis. From this, they extracted features from the
samples using the PEView tool. The malware files were randomly taken from various sources, and the legitimate files were taken from Windows and Programs Files folder.

Yuval Elovic, Chanan Gleze, and Robert Moskovitch [13] have extracted two types of static features, N-grams, and Win32 executables Portable Executable Header. They have implemented a tool that extracted 5-grams from the binary representation of a file. In the same manner, Mozammel Chowdhury, Azizur Rahman, and Md Rafiqul Islam [14] proposed a work in which they introduce data mining. In this work, they also extracted features by analyzing executable files based on both static and dynamic analysis. They extracted two types of features from the executable's files, N-gram features, and Windows API calls on which classifiers were trained.

3. Proposed Methodology

There are some critical steps that need to be taken into consideration when solving a Machine Learning problem. Hence, in this thesis work, we have considered four major steps to implement our framework.

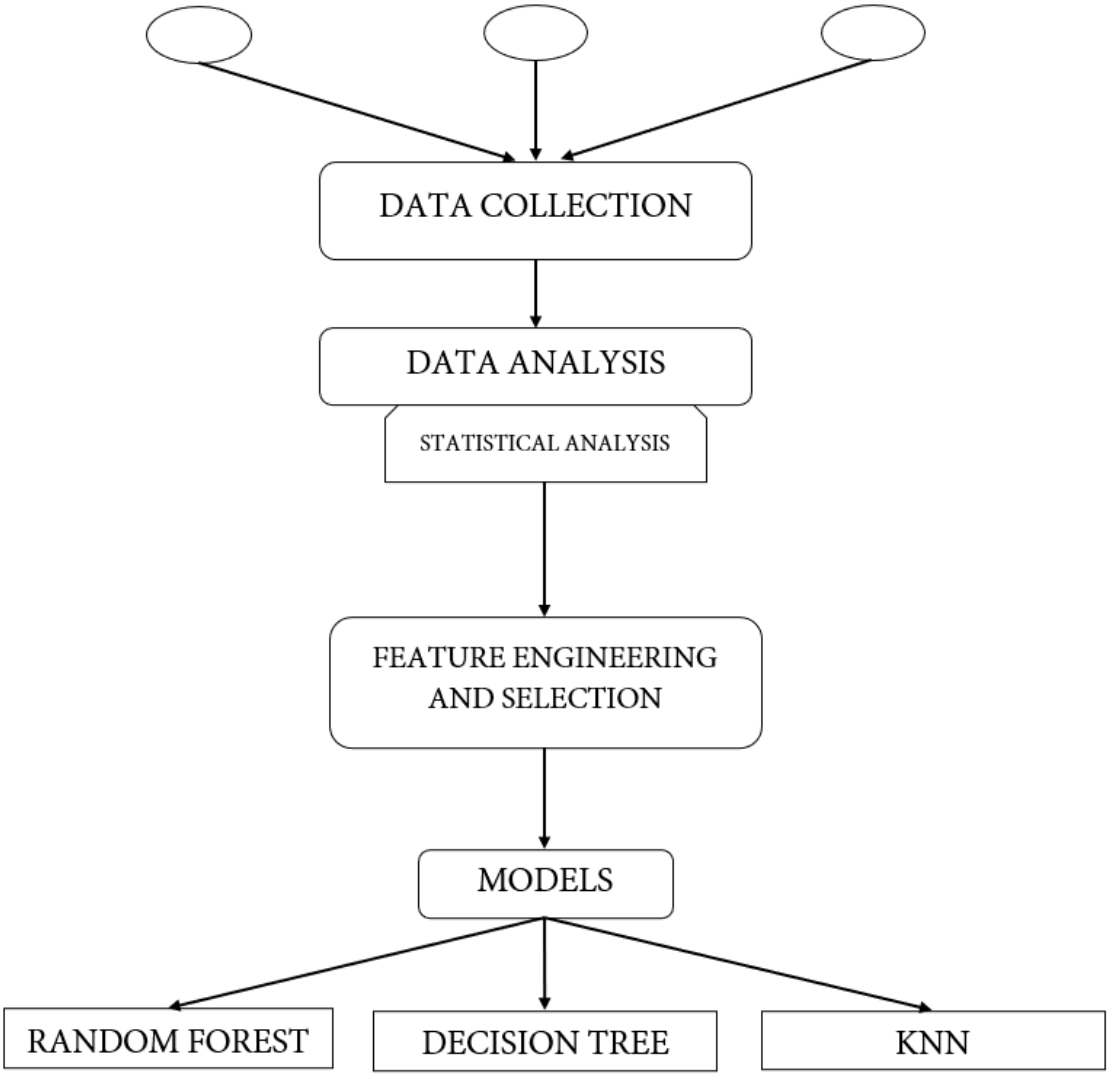

Figure 3.1: Experiment methodology

3.1 Data Collection and Description

The dataset used in this work was taken from Kaggle [15]. It is part of a competition hosted by Microsoft in 2018. In the dataset, each row corresponds to a machine uniquely identified by a Machine Identifier. The dataset consists of 8M records and 83 columns. To allow the data to be loaded into our computer's memory, we considered only a subset of the data. The final data to train the models consisted of 567730 records.

3.2 Data Analysis

In Machine Learning, the first critical step before building the models is to understand the data. We can do that by performing some statistical analysis, and also by visualizing the distribution of the data for different attributes. We realized that the dataset was highly biased. In the dataset, columns like 'PuaMode', and 'Census_ProcessorClass' have more than 90% missing values. We, therefore, set a threshold value of 30% for evaluation. All the columns with more than 30% missing values have to be deleted. These columns will not be useful to our classifiers as most of the values are missing. We also have some columns with different values in each row. Having these columns would only confuse our models. Therefore, the best solution would be to ignore them from the dataset. In the end, we will have columns with only a few missing values that will be handled later.

3.3 Feature Engineering and Selection

After getting a basic understanding of the data, our next goal is to make feature engineering, and by selecting relevant features to build the models. We have to improve the quality of the data. The dataset consists of 56,7730 records with 482,571 malware and 85,159 legitimate files. We see that the data is highly imbalanced. Training a classifier on this kind of data might lead to an unexpected result. In Machine Learning, there are many ways to deal with this kind of problem. In our experiment, we went on with the over-sampling method. RandomOverSampler [16] is a module that is available in python that helps us to deal with the imbalance problem. It is a technique that will randomly select data in

the minority class and duplicate them to balance the data. We ended up with perfectly balanced data. The next step consisted of selected relevant features. By using the seaborn library, we plotted the correlation between each feature using the heatmap[17] available in the seaborn library to understand how much the features were correlated to each other. The dataset consists of 83 columns, so plotting all these columns at the same time will make visualization difficult. Thus for the first time, we plotted only a subset of the column list from column 1 to the 15th column.

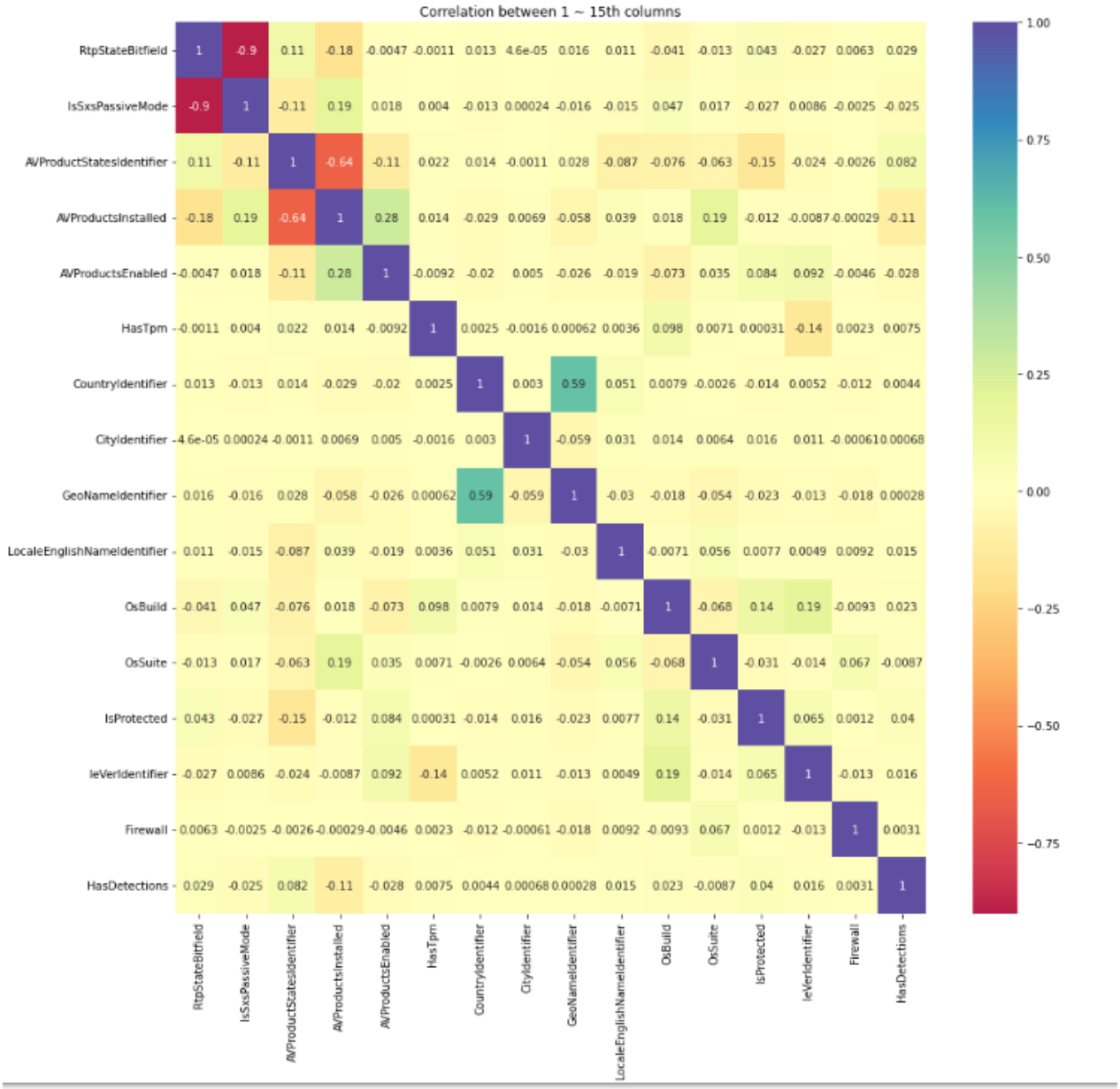

Figure 3.3.1: Matrix correlation between features

For example, the correlation between ‘IsSxsPassiveMode’ and ‘RtpStateBitfield’ is high. We can delete one column and keep the other one. We deleted the column RtpStateBitfield and kept the other one.

We have to be very careful when deciding what column to delete. For that, we have to understand what each column contains. We can do that by visualizing each column.

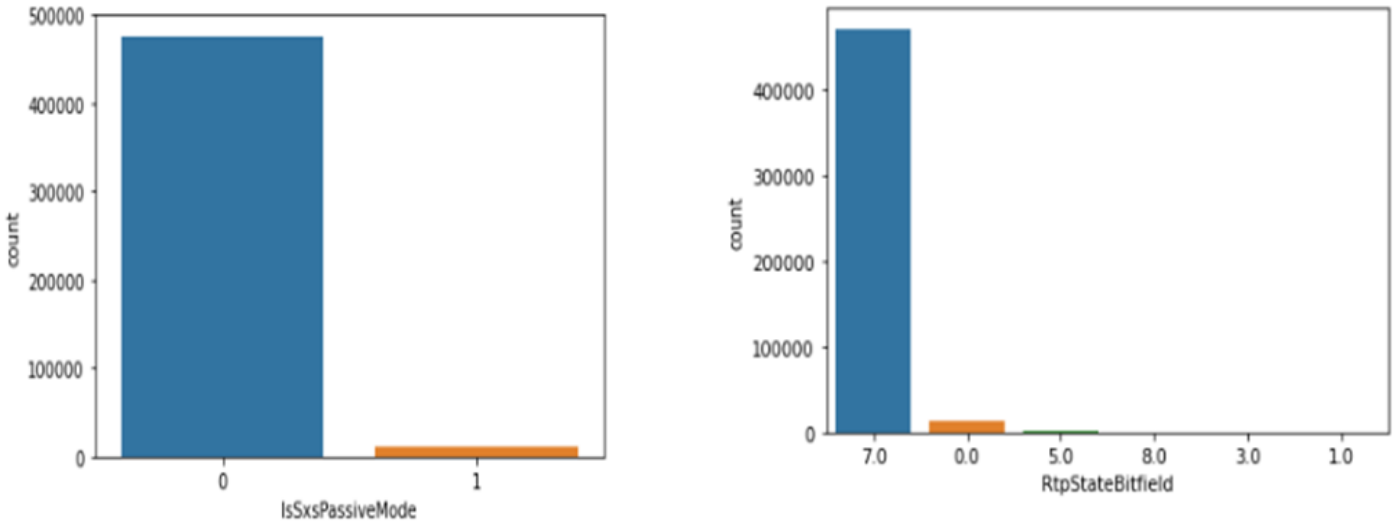

Figure 3.3.2: Distribution of data

The column RtpStateBitfield contains six different types of values. From the figure above, It is clear that the values in this columns were not equally distributed. Then we deleted it and kept the IsSxsPassiveMode column, which has two types of value. For the next phase, we plotted from the 15th column to the 30th column. We did the same process to select all the relevant features. In the end, we were left with 58 features out of 83 completely uncorrelated. We normalized the data using the StandardScaler module. This allows the data to be on the same scale before training.

3.4 Models

Once we have ameliorated the quality of the data, normalized the data, and selected relevant features, we now have to build the models. In this work, we built only three classifiers. During the training phase we split the data into training and testing data. We took 80% of the data for the training phase, and 20% for the testing phase. Finally, we trained our models and tested them on the test data. Before building the models, let's understand how each classifier works behind the scenes to make predictions.

3.4.1 Random Forest and Decision Tree

Random Forest [18] and Decision Tree [19] are two types of supervised learning approaches that can be used in both classification and regression problems. Random Forest is a tree-based classifier that uses the technique of ensemble learning, which consists of combining multiple classifiers into one to solve more complex problems. During training, Random Forest builds multiple internal decision trees that each will give an output. From these outputs, Random Forest chooses the final decision based on the Majority voting for the given problem.

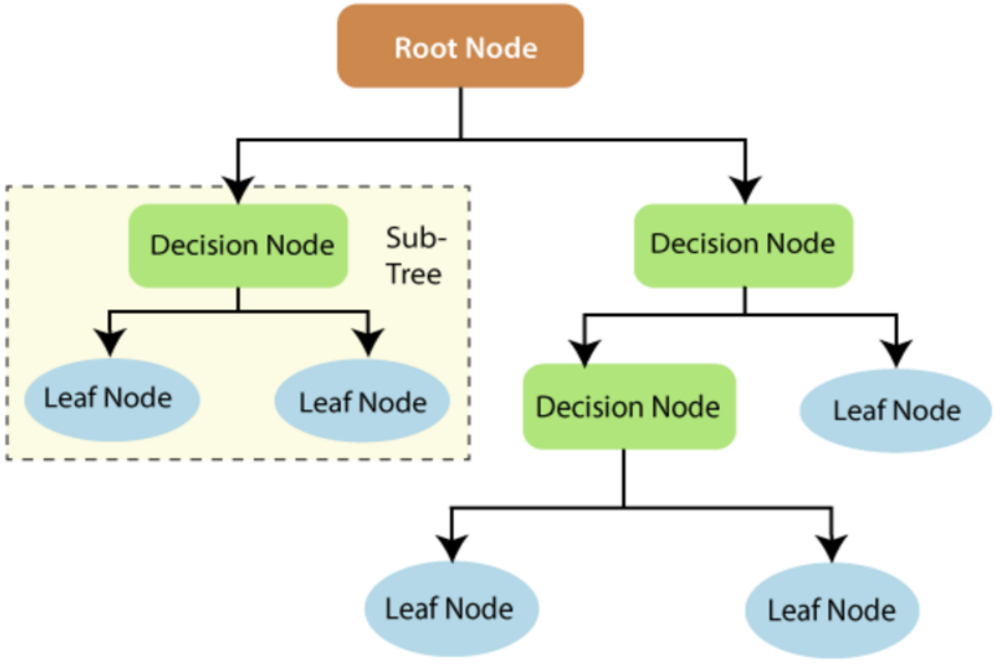

Figure 3.4.1.1: Construction of decision tree

When constructing individual trees, Random Forest will not consider all features. Each tree will be different from one another. This leads to feature space reduction.

In the Decision Tree, each node denotes a test based on an attribute, each branch represents the result of the test, and each leaf contains the class label.

Decision Tree makes classification by sorting the instances down the tree from the root to some terminal node, which provides the classification of the instance. This process keeps happening recursively for every subtree rooted at the new node.

The two algorithms have many parameters that can be tuned to improve the accuracy of a classification problem.

A. Random Forest Model

As stated above, Random Forest has many parameters that can be used to train the model to allow better accuracy. In this experiment, we tried a set of values for each parameter and considered only the parameters that gave us the best results.

The training phase was done as follows:

We trained the model with the default values for each parameter, and we got an accuracy of 98.4%, an FPR value of 0.0233, and an f1-score value of 0.98 for both malware and legitimate files. Then, We tuned the parameters to see if we could get better results. The parameters we tuned are n_estimators, max_features, min_samples_split, min_samples_leaf, and bootstrap.

In the end, we were able to get a better result than the previous one. We got the optimal result with the values shown below:

```
rf_model=
ek.RandomForestClassifier(n_estimators=400,max_features='auto',max_depth=6
0,min_samples_split=2,min_samples_leaf=1,bootstrap=True)

rf_model.fit(X_train,y_train)

print (f'Train Accuracy - : {rf_model.score(X_train,y_train):.3f}')

print (f'Test Accuracy - : {rf_model.score(X_test,y_test):.3f}')
```

The n_estimators parameter defines the number of trees we want to build before taking the majority vote. The max_features defines the maximum number of features Random Forest is allowed to try in the individual tree. When we want to control what depth the trees should grow, then the max_depth parameter is tuned for this purpose. The min_samples_split and the min_samples_leaf attributes respectively define the minimum number of features required to split an internal node, and the minimum number of samples to be at the leaf node.

With these values, we got an accuracy of 98.5%, an FPR value of 0.0208, and an f1-score of 99% for both malware and legitimate files.

B. Decision Tree Model

We did the same thing during training for the Decision Tree as well. After training the model with the default values, we used different values for each parameter to increase the accuracy. After the experiment, we noticed that the results achieved with the default values were the best ones.

```
dt_model = DecisionTreeClassifier()

dt_model.fit(X_train,y_train)

print (f'Train Accuracy - : {dt_model.score(X_train,y_train):.3f}')

print (f'Test Accuracy - : {dt_model.score(X_test,y_test):.3f}')
```

We got an accuracy of 0.906 with an FPR value of 0.177, and an F1-score of 91% for both malware and legitimate files.

3.4.2 K-Nearest Neighbors

KNN [20], like Random Forest and Decision Tree, is a supervised learning approach. KNN tries to predict the label of a data point by calculating the distance between the data point and all the training data.

Many distance metrics can be used such as Euclidean Distance, and Manhattan Distance.

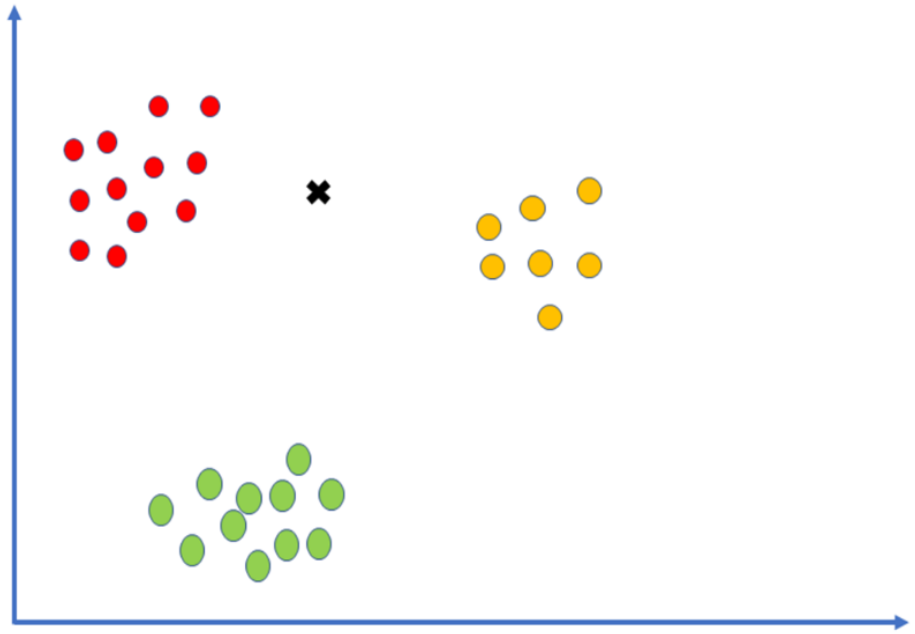

Figure 3.4.2.1: KNN

The K-value gives us the number of nearest points to consider for the new data point. After calculating the distance between the data point and all the training data points. We classify the new data point as the category in which the data are close to it. In the KNN part, we tuned only the k parameter. After testing different values of K, we realized that the default K value gave us the best result.

```
knn_model = KNeighborsClassifier()

knn_model.fit(X_train,y_train)

print (f'Train Accuracy - : {knn_model.score(X_train,y_train):.3f}')

print (f'Test Accuracy - : {knn_model.score(X_test,y_test):.3f}')
```

4. Results Analysis

We ran a different experiment for each model with different parameter values. We will only show a few parameters with their values along with their respective FPR values in this section. We trained the models in many phases. For the sake of simplicity, we chose only the three phases with the higher results.

	n_estimators	max_features	max_depth	min_samples_split	min_samples_leaf	bootstrap	FPR
Phase1	100	auto	None	2	1	True	0.0233
Phase2	400	auto	60	2	1	True	0.0208
Phase3	250	sqrt	20	5	2	False	0.277

Random forest

	max_depth	max_features	criterion	min_samples_split	min_samples_leaf	splitter	FPR
Phase1	None	None	gini	2	1	best	0.177
Phase2	4	sqrt	entropy	4	10	random	0.607
Phase3	8	log2	entropy	8	10	best	0.434

Decision Tree

	K	FPR
Phase1	5	0.395
Phase2	10	0.374
Phase3	20	0.409

KNN

At the end of the entire experiment, we only kept the parameters that gave us the best result for each model. The table below shows the summary of the final result.

	TPR	FPR	Accuracy	AUC
Decision Tree	0.993	0.177	0.907	0.908
Random Forest	0.992	0.0208	0.985	0.997
K-Nearest Neighbors	0.916	0.395	0.759	0.846

In this work, our goal was to achieve an FPR value close to 0, and we can clearly see that from the values above, Random Forest gives us the best result with an FPR value of 0.0208 and an f1-score of 0.99 for both malware and legitimate files. The precision value gives us the ratio of correctly classified as positive to the total classified positive observations, the recall value gives us the ratio of correctly classified to all observations in the actual class. Finally, the f1-score takes into consideration both precision and recall. It gives us better intuition about how good our classifiers are.

5. Conclusion and Future Work

The biggest challenge in the world of IT systems is security. Over the years, many techniques have emerged from researchers. Machine Learning is used to overcome the problem of traditional anti-virus systems. In this experiment, we used three classifiers to detect malware, and we got the best results with Random Forest. In the near future, we plan to introduce more complex classification algorithms. Collecting the most recent samples of malware and legitimate files might help to get better results, and doing intensive feature engineering to ameliorate the quality of the data, and select the most important features will surely give better results. Finally, making more hyperparameters tuning might lead to better results.

References

[1] AV-TEST Institute, https://www.av-test.org/en/statistics/malware/

[2] Simon Kramer, Julian C. Bradfield,“A general definition of malware”.

[3] https://www.malwarebytes.com/malware.

[4] James Scott, “Signature Based Malware Detection is Dead”,2017.

[5] Andreas Moser, Christopher Kruegel, and Engin Kirda, “Limits of status Analysis for Malware Detection”.

[6] Amir Afianian, Salman Niksefat, and Babak Sadeghiyan, “Malware Dynamic Analysis Evasion Techniques : a survey”, 2019.

[7] Zane Markel and Michael Bilzor, “Building a Machine Learning Classifier for Malware Detection”, 2014.

[8] Junho Choi, Hayoung Kim, Chang Choi, and Pankoo Kim, “Efficient Malicious Code Detection Using N-Gram Analysis and SVM”.

[9] Zhongru Wang, Peixin Cong, and Weiqiang Yu, “Malicious Code Detection and Technology Based on Metadata Machine Learning”,2020.

[10] Ivan Firdausi, Charles Lim, Alva Erwin, and Anto Satriyo Nugroho, “Analysis of Machine Learning used in behavior-based Malware Detection”.

[11] Athiq Reheman Mohammed (&), G. Sai Viswanath, K. Sai babu, and T. Anuradha, “Malware Detection in Executable Files Using Machine Learning”.

[12] Nur Syuhada Selamat, Fakariah Hani Mohd Ali, “Comparison of malware detection techniques using machine learning algorithm”.

[13] Yuval Elovic, Chanan Gleze and Robert Moskovitch, “Applying Machine Learning Techniques for Detection of Malicious code in network traffic”.

[14] Mozammed Chowdhury, Azizur Rahman, Md Rafiqul Islam, “Malware Analysis and Detection using Data Mining and Machine Learning Classification”.

[15] https://www.kaggle.com/c/microsoft-malware-prediction/overview

[16] Roweida Mohammed, Jumanah Rawashdeh and Malak Abdullah, “Machine Learning with Oversampling and Undersampling Techniques: Overview study and Experimental Results”,2020.

[17] https://seaborn.pydata.org/generated/seaborn.heatmap.html

[18] Yanli Liu, Yourong Wang, and Jian Zhang, “New Machine Learning Algorithm: Random Forest”, 2012.

[19] Arundhati Navada, Aamir Nizam Ansari, Siddharth Patil, Balwant A. Sonkamble, “Overview of use of Decision Tree algorithms in Machine Learning”, 2011.

[20] Lishan Wang, “Research and Implementation of Machine Learning Classifier based on KNN”, 2019.